CONTENTS

6 Model Fitting

6.1 Linear Regression With Python

6.1.1 Simple and Multiple Regression

Step 1: Import packages

First you need to do some imports. In addition to numpy, you need to import statsmodels.api:

```
import numpy as np
import statsmodels.api as sm
from sklearn.preprocessing import PolynomialFeatures
```

Now you have the packages you need.

Step 2: Provide data and transform inputs

The second step is defining data to work with. The regressors,x and predictor, y should be arrays.

```
y = np.array([5, 20, 14, 32, 22, 38])
n = len(y)
x0 = np.ones(n).reshape((-1, 1))
```

```
x1 = np.array([5, 15, 25, 35, 45, 55]).reshape((-1, 1))
x = np.hstack((x0, x1))
```

Now, you have two arrays: the x and y. You should call .reshape() on x because this array is required to be two-dimensional, or to be more precise, to have one column and as many rows as necessary. That's exactly what the argument (-1, 1) of .reshape() specifies.

Step 3: Create a model and fit it

The regression model based on ordinary least squares is an instance of the class

statsmodels.regression.linear_model.OLS.

This is how you can obtain one:

```
model = sm.OLS(y, x)
```

Once your model is created, you can apply .fit() on it:

```
results = model.fit()
```

By calling .fit(), you obtain the variable results, which is an instance of the class statsmodels.regression.

linear_model.RegressionResultsWrapper.

This object holds a lot of information about the regression model.

Step 4: Get results

You can call .summary() to get the table with the results of linear regression:

```
print(results.summary())
```

```
 OLS Regression Results
==============================================================================
Dep. Variable:            y   R-squared:                       0.716
Model:                  OLS   Adj. R-squared:                  0.645
Method:      Least Squares   F-statistic:                     10.08
Date:       Fri, 29 Oct 2021   Prob (F-statistic):             0.0337
Time:              17:19:23   Log-Likelihood:                -19.071
No. Observations:        6   AIC:                             42.14
Df Residuals:            4   BIC:                             41.73
Df Model:                1
Covariance Type:            nonrobust
==============================================================================
```

```
        coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const  5.6333      5.872      0.959      0.392     -10.670      21.936
x1     0.5400      0.170      3.175      0.034       0.068       1.012
==============================================================================
Omnibus:                        nan   Durbin-Watson:                   3.606
Prob(Omnibus):                  nan   Jarque-Bera (JB):                0.651
Skew:                         0.008   Prob(JB):                        0.722
Kurtosis:                     1.387   Cond. No.                         69.8
==============================================================================
```

Step 5: Predict response

You can obtain the predicted response on the input values used for creating the model using .fittedvalues or .predict() with the input array as the argument:

```
print('predicted response:',results.fittedvalues)
print('predicted response:',results.predict(x))
```

```
predicted response: [ 8.33333333 13.73333333 19.13333333 24.53333333
 29.93333333 35.33333333]
```

This is the predicted response for known inputs. If you want predictions with new regressors, you can also apply .predict() with new data as the argument:

```
x_new = np.array([1, 20])
y_new = results.predict(x_new)
print("y_new=",y_new)
```

6.1.2 Testing for significance of regression

Is the regression equation that uses information provided by the predictor variables $x_1, x_2, \ldots, x_k$ substantially better than the simple predictor y that does not rely on any of the $x-$values? This question is answered using an overall F test.

1. The hypotheses are $H_0 : \beta_1 = \cdots = \beta_k = 0$ versus H_1 : At least on β's $\neq 0$
2. The test statistic is F
3. Find the p-value
4. Reject H_0 if p-value $< \alpha$, where α is the level of significance, usually is 5%.

When the null hypothesis is rejected, this means that that the model using $x_1, x_2, \cdots, x_k$ as predictor variables is useful for y.

Example 1.

Consider the data shown below:

y	x	y	x
41	14	34	11
28	8	30	9
30	9	19	4
25	7	41	14
36	12	28	8

You fit the above data to $Y = \beta_0 + \beta_1 X + \epsilon$.

(a) State the estmated regression function.

(b) Obtain a prediction for a new observation y_h when $x_h = 17.0$.

Sol:

(a) $\hat{y} = 10.1947 + 2.1881x$

(b) $y_h = 10.1947 + 2.1881(17.0) = 47.3916$

```
Python code:
import numpy as np
import statsmodels.api as sm
y = np.array([41,28,30,25,36,19,41,283430])
n = len(y)
x0 = np.ones(n).reshape((-1,1))
```

```
x1 = np.array([14,8,9,7,12,4,14,8119]).reshape((-1,1))
x = np.hstack((x0,x1))
model = sm.OLS(y,x)
results = model.fit()
beta = [10.1947 2.1881]
print("Beta hat = ",beta)
#print(results.summary())
xnew = np.array([1,17.0])
ynew = results.predict(17.0)
print("y hat = ",ynew)
Output:
Beta hat = [10.19469026548672 2.188053097345133]
y_new= [[47.39159292]]
```

Example 2.

The data file VegiB.csv record a flavor and texture score, Y (between 0 and 100) for 12 brands of meatless hamburgers along with the price, X_1, number of calories, X_2, amount of fat, X_3, and amount of sodium per burger, X_4. Assuming that regression model $Y = \beta_0+\beta_1 X_1+\beta_2 X_2+\beta_3 X_3+\beta_4 X_4 + \epsilon$ is appropriate.

(a) Fit a regression model to the data. State the estimated regression function.

(b) Determine the predicted value for the mean score of flavor and texture for all brands of meatless hamburgers with the price = 89, number of calories = 155, amount of fat = 4, and amount of sodium per burger = 498.

(c) Is the model $Y = \beta_0+\beta_1X_1+\beta_2X_2+\beta_3X_3+\beta_4X_4+\epsilon$ useful for estimating the texture score?

Sol:

(a) $\hat{Y} = 59.8488 + 0.1287X_1 + -0.5805X_2 + 8.4982X_3 + 0.0488X_4$

(b) $\hat{Y}_h = 59.8488 + 0.1287(89) + -0.5805(155) + 8.4982(4)+0.0488(498) = 39.603738193072914$

(c) since p-value $= 0.24434282165926965 > 0.05$. The null hypothesis of the model not significance for estimating the texture score is not rejected. Hence the model $Y = \beta_0 + \beta_1X_1 + \beta_2X_2+\beta_3X_3+\beta_4X_4+\epsilon$ is not significant for estimating the texture.

```
Python code:
import pandas as pd
import numpy as np
import statsmodels.api as sm
import matplotlib.pyplot as plt
df = pd.read_csv('VegieB.csv')
y = df.y
n = len(y)
x0 = np.ones(n).reshape(-1,1)
x1 = np.array(df.x1).reshape(-1,1)
```

```
x2 = np.array(df.x2).reshape(-1,1)
x3 = np.array(df.x3).reshape(-1,1)
x4 = np.array(df.x4).reshape(-1,1)
x = np.hstack((x0,x1,x2,x3,x4))
x, y = np.array(x), np.array(y)
model = sm.OLS(y, x)
results = model.fit()
Beta = results.params
print("beta = ",Beta)
x_new = np.array([1,89,155,4,498]).reshape(-1,5)
print("x_new=",x_new)
y_new = results.predict(x_new)
print("y_new=",y_new)
print(results.summary())
Output:
beta =  [59.8488 0.1287 -0.5805 8.4982  0.0488 ]
x_new= [[1  89  155 4   498]]
y_new= [39.603738193072914]
```

6.1.3 Testing the Significance of the Partial Regression Coefficients

Once you have determined that the model is useful for predicting y, you should explore the nature of the "usefulness" in more detail. Do all the predictor variables add important information for prediction in the presence of other predictors already in the model?

1. The hypotheses are $H_0 : \beta_j = 0$ versus $H_1 : \beta_j \neq 0$ for $j = 1, 2, \ldots, k$
2. The test statistics is $t = \dfrac{\hat{\beta}_j}{SE(\hat{\beta}_j)}$, where $SE(\hat{\beta}_j)$ is the standard error of $\hat{\beta}_j$
3. Find the p value
4. Reject H_0 if p-value $< \alpha$.

Example 3.

Consider the data below:

y	x_1	x_2	x_3	x_4
280.3	7.9	31.6	15.4	30.2
188.0	4.6	29.4	8.7	16.8
191.4	4.7	29.5	8.9	17.3
169.3	3.9	28.9	7.3	14.1
249.2	6.8	30.9	13.1	25.7
118.3	2.0	27.7	3.6	6.7
285.6	8.1	31.7	15.7	31.0
181.8	4.4	29.2	8.2	15.9
226.6	6.0	30.3	11.5	22.4
203.1	5.1	29.8	9.8	19.0

Assuming that regression model $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_4 + \epsilon$ is appropriate.

(a) Find the test statistic for testing $H_0 : \beta_1 = \beta_2 = \beta_3 = \beta_4 = 0$.

(b) Find the p-value for testing $H_0 : \beta_1 = \beta_2 = \beta_3 = \beta_4 = 0$.

(c) Find the test statistic for testing $H_0 : \beta_1 = 0$.

(d) Find the p value for testing $H_0 : \beta_1 = 0$.

Sol:

(a) F = 5596015.587982019

(b) p-value = 8.25363912287115e-17

(c) t = 3.016624174704267

(d) p-value$_1$ = 0.029530395921302244

```
import numpy as np
import statsmodels.api as sm
y = [280.3, 188.0, 191.4, 169.3, 249.2,
118.3, 285.6, 181.8, 226.6,203.1]
n = len(y)
x0 = np.array(np.repeat(1,n)).reshape(-1,1)
x1 = np.array([7.9, 4.6, 4.7, 3.9, 6.8,
2.0, 8.1, 4.4, 6.0, 5.1]).reshape(-1,1)
x2 = np.array([31.6, 29.4, 29.5, 28.9, 30.9,
27.7, 31.7, 29.2, 30.3, 29.8]).reshape(-1,1)
x3 = np.array([15.4, 8.7, 8.9, 7.3, 13.1,
3.6, 15.7, 8.2, 11.5, 9.8]).reshape(-1,1)
x4 = np.array([30.2, 16.8, 17.3, 14.1, 25.7,
6.7, 31.0, 15.9, 22.4, 19.0]).reshape(-1,1)
x = np.hstack((x0, x1,x2,x3,x4))
x, y = np.array(x), np.array(y)
```

```
model = sm.OLS(y, x)
results = model.fit()
Fstat = results.fvalue
pvalue = results.f_pvalue
Tstat = results.tvalues[1]
pvaluej = results.pvalues[1]
print("F=", Fstat)
print("P-value", pvalue)
print("T Stat", Tstat)
print("P-value_1", pvaluej)
```

6.1.4 Polynomial Regression

Implementing polynomial regression is very similar to linear regression. There is only one extra step: you need to transform the array of inputs to include non-linear terms such as x^2.

Step 1 Import packages and classes

In addition to numpy and statsmodels, you should also import the class

PolynomialFeatures from sklearn.preprocessing:

```
import numpy as np
import statsmodels.api as sm
from sklearn.preprocessing import PolynomialFeatures
```

Step 2 Provide data and transform input data

The input and output and is the same as in the case of linear regression:

```
x = np.array([5, 15, 25, 35, 45, 55]).reshape((-1, 1))
y = np.array([15, 11, 2, 8, 25, 32])
```

As you've seen earlier, you need to include x^2 (and perhaps other terms) as additional features when implementing polynomial regression. For that reason, you should transform the input array x to contain the additional column(s) with the values of x^2 (and eventually more features). It's possible to transform the input array in several ways (like using insert() from numpy), but the class PolynomialFeatures is very convenient for this purpose. Let's create an instance of this class:

```
transformer = PolynomialFeatures(degree=2,
                                 include\_bias=False)
```

The variable transformer refers to an instance of PolynomialFeatures which you can use to transform the input x.

You can provide several optional parameters to PolynomialFeatures:

- degree is an integer (2 by default) that represents the degree of the polynomial regression function.
- interaction_only is a Boolean (False by default) that decides whether to include only interaction features (True) or all features (False).
- include_bias is a Boolean (True by default) that decides whether to include the bias (intercept) column of ones (True) or not (False).

Before applying transformer, you need to fit it with .fit():

```
transformer.fit(x)
```

Once transformer is fitted, it's ready to create a new, modified input. You apply .transform() to do that:

```
x_ = transformer.transform(x)
```

You can also use .fit_transform() to replace the three previous statements with only one:

```
x_ = PolynomialFeatures(degree=2,
                        include_bias=False).fit_transform(x)
```

That's fitting and transforming the input array in one statement with .fit_transform().

Example 4.

Consider the data shown below:

y	x	y	x
1,101,624	32	412,679	25
170,115	20	248,344	22
170,115	20	970,865	31
89,304	17	22,524	12
647,589	28	206,462	21
10,980	10	206,462	21
1,245,168	33	744,593	29
138,793	19	89,304	17

Fit the polynomial regression models up to order 4, then answer the following questions:

(a) Fill in the table below:

Polynomial model	Adjusted R^2	AIC	BIC
First order			
Second order			
Third order			
Fourth order			

(b) Determine a model that best fit the data based on the adjusted R^2.

(c) Write down the equation of the best fitted curve that you have selected from part (b).

Sol:

(a)

Polynomial model	Adjusted R^2	AIC	BIC
First order	0.861	428.9	430.5
Second order	0.996	372.0	374.3
Third order	1.0	287.1	290.2
Fourth order	1.0	0.527	4.39

(b) Since the fourth-order polynomial has the largest adjusted R^2 thus, model with polynomial fourth degree should be selected.

(c) $y = -10.5229 + 2.76x + 0.0648x^2 + 0.6574x^3 + 1.0299x^4$

```
import numpy as np
from sklearn.preprocessing import PolynomialFeatures
import statsmodels.api as sm
y = np.array([1,101,624,170,115,170,115,89,304,647,589,10,980,1,245,168,1
              412,679,248,344,970,865,22,524,206,462,206,462,744,593,89,3
print(y)
x = np.array([32,20,20,17,28,10,33,19,
              25,22,31,12,21,21,29,17]]).reshape((-1,1))
x1st = PolynomialFeatures(degree=1, include_bias=True).fit_transform(x)
x2nd = PolynomialFeatures(degree=2, include_bias=True).fit_transform(x)
x3rd = PolynomialFeatures(degree=3, include_bias=True).fit_transform(x)
x4th = PolynomialFeatures(degree=4, include_bias=True).fit_transform(x)
model1 = sm.OLS(y, x1st)
results1 = model1.fit()
model2 = sm.OLS(y, x2nd)
results2 = model2.fit()
model3 = sm.OLS(y, x3rd)
results3 = model3.fit()
model4 = sm.OLS(y, x4th)
results4 = model4.fit()
print("Adj. R-squared model1 = ", results1.rsquared_adj)
print("AIC model1 = ", results1.aic)
print("BIC model1 = ", results1.bic)
print("Adj. R-squared model2 = ", results2.rsquared_adj)
print("AIC model2 = ", results2.aic)
print("BIC model2 = ", results2.bic)
print("Adj. R-squared model3 = ", results3.rsquared_adj)
print("AIC model3 = ", results3.aic)
print("BIC model3 = ", results3.bic)
print("Adj. R-squared model4 = ", results4.rsquared_adj)
print("AIC model4 = ", results4.aic)
print("BIC model4 = ", results4.bic)
```

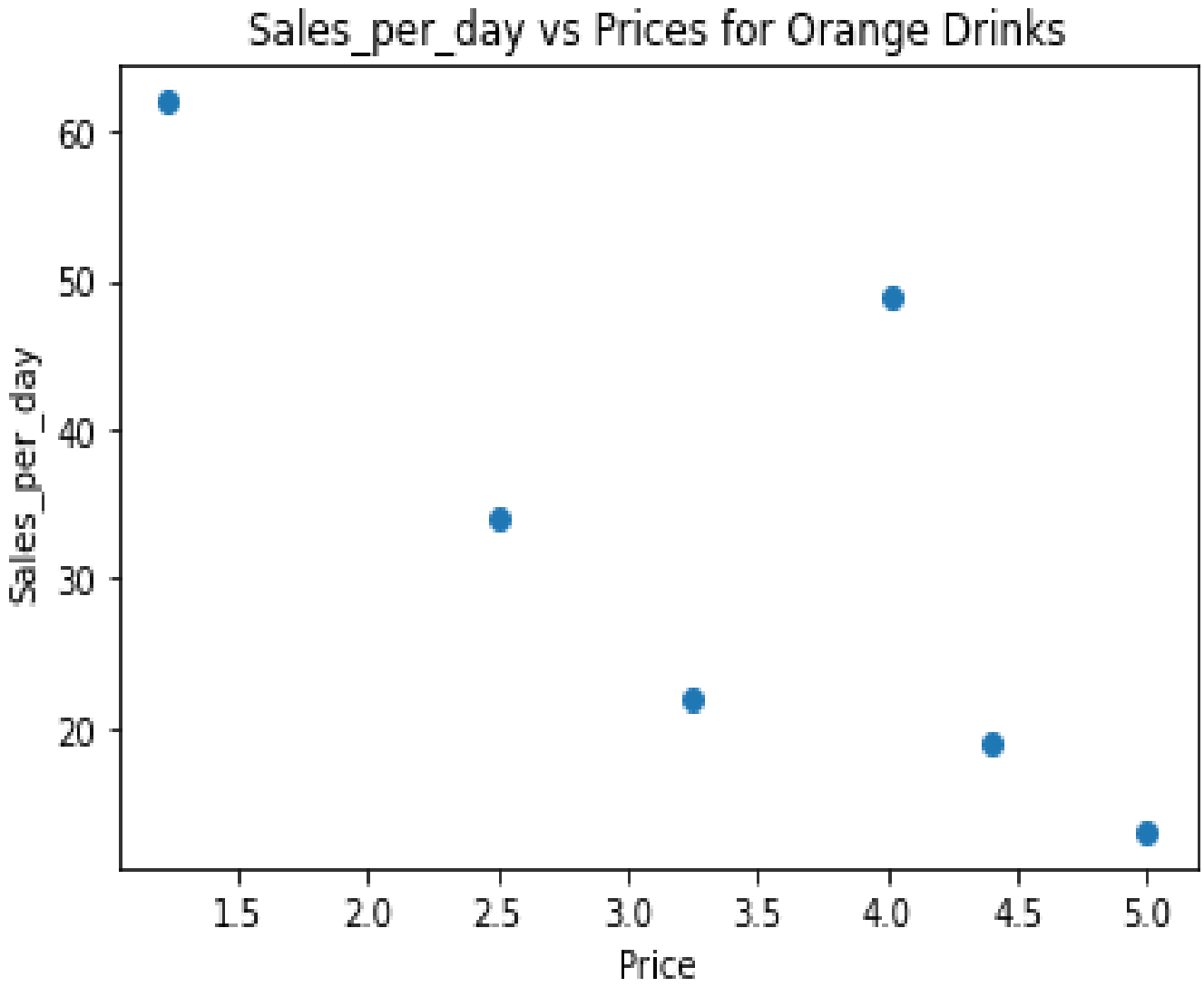
```
print("Beta hat = ", results4.params)
#print("results1",results1.summary())
#print("Results2",results2.summary())
#print("Results3",results3.summary())
#print("results4",results4.summary())
Output:
Adj. R-squared model1 =  0.861
AIC model1 =  428.9
BIC model1 =  430.5
Adj. R-squared model2 =  0.996
AIC model2 =  372.0
BIC model2 =  374.3
Adj. R-squared model3 =  1.0
AIC model3 =  287.1
BIC model3 =  290.2
Adj. R-squared model4 =  1.0
AIC model4 =  0.527
BIC model4 =  4.39
Beta hat =  [-10.5229 2.7560 0.0648 0.6574 1.0299]
```

6.2 Visualizing Data in Python

A scatter plot is a visual representation of how two variables relate to each other. You can use scatter plots to explore the relationship between two variables, for example by looking for any correlation between them.
One of the most popular modules is **Matplotlib** and its submodule **pyplot**, often referred to using the alias **plt**. Matplotlib provides a very versatile tool called **plt.scatter()** that allows you to create both basic and more complex scatter plots.
A cafe sells six different types of bottled orange drinks. The owner wants to understand the relationship between the price of the drinks and how many of each one he sells, so he keeps track of how many of each drink he sells every day. You can visualize this relationship as follows:

```
import matplotlib.pyplot as plt
price = [2.50, 1.23, 4.02, 3.25, 5.00, 4.40]
sales_per_day = [34, 62, 49, 22, 13, 19]
```

```
plt.scatter(price, sales_per_day)
plt.title("Sales_per_day vs Prices for Orange Drinks")
plt.xlabel("Price")
plt.ylabel("Sales_per_day")
plt.show()
```



Example 5.

The quality of orange juice produced by a manufacturer is constantly monitored. Is there a relationship between the sweetness index and chemical measure such as the amount of water soluble pectin (parts per million) in the orange juice? Data collected on these two variables for 24 production runs at juice manufacturing plant are save in orjuice.csv file. Suppose a manufacturer wants to use simple linear regression to predict the sweetness(Y) from the amount of pectin (X).

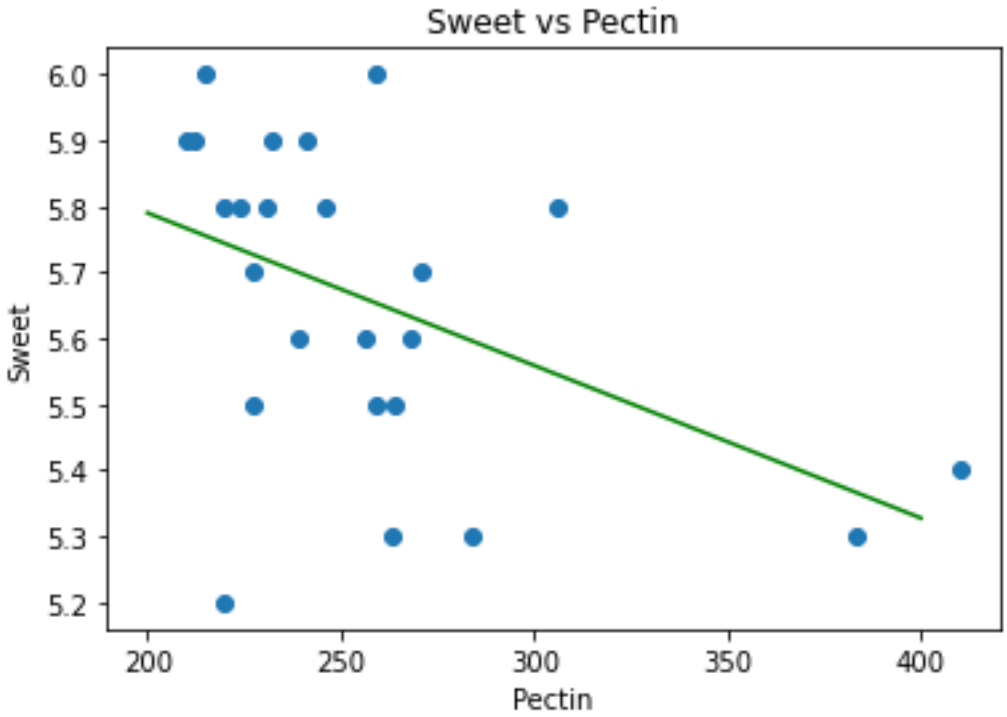
(a) Fit the data to the model $y = \beta_0 + \beta_1 x + \epsilon$.

(b) Plot the fitted regression function and the data.

Sol:

(a) $\hat{y} = 6.252067909048172 - 0.0023106258824640955x$

(b)




```
import pandas as pd
import numpy as np
import statsmodels.api as sm
import matplotlib.pyplot as plt
df = pd.read_csv('orjuice.csv')
#print(df)
model1 = np.poly1d(np.polyfit(df.Pectin, df.Sweet, 1))
print(model1)
plt.scatter(df.Pectin, df.Sweet)
plt.title("Sweet vs Pectin")
plt.xlabel("Pectin")
plt.ylabel("Sweet")
polyline = np.linspace(200, 400, 500)
plt.plot(polyline, model1(polyline), color='green')
```

Example 6.

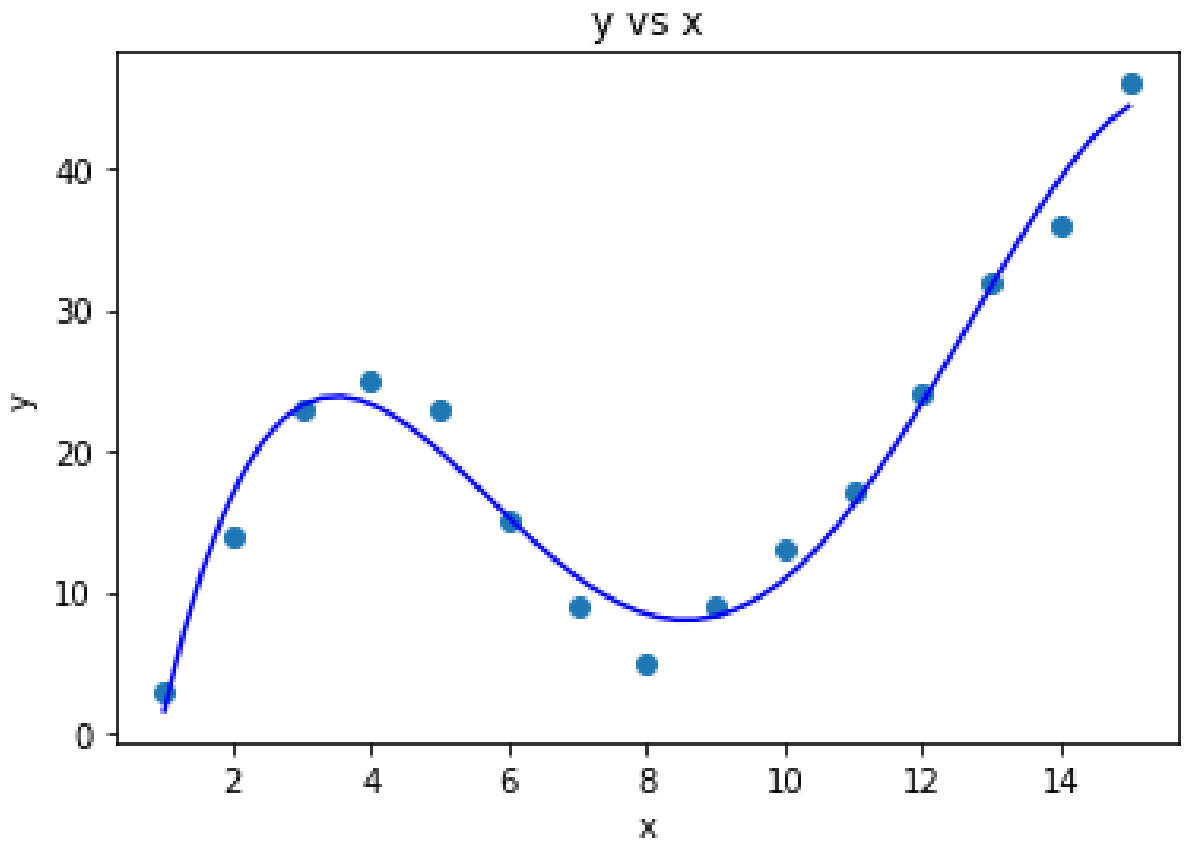
You are given the following data:

x	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
y	3	14	23	25	23	15	9	5	9	13	17	24	32	36	46

Fit the data to the polynomial regression model of degree 4. Created a scatter plot of the data and add the fitted polynomial line to scatterplot.

Sol:

```
import numpy as np
import matplotlib.pyplot as plt
x = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
y = [3, 14, 23, 25, 23, 15, 9, 5, 9, 13, 17, 24, 32, 36, 46]
#fit polynomial models up to degree 5
model4 = np.poly1d(np.polyfit(x, y, 4))
#create scatterplot
polyline = np.linspace(1, 15, 50)
plt.scatter(x, y)
plt.title("y vs x")
plt.xlabel("x")
plt.ylabel("y")
#add fitted polynomial lines to scatterplot
plt.plot(polyline, model4(polyline), color='blue')
plt.show()
```


y vs x
y
x
0
10
20
30
40
2
4
6
8
10
12
14

Example 7.

Consider the data shown below:

y,	x	y	x
1030	11.2	633	10.1
429	9.3	498	9.6
429	9.3	987	11.1
348	8.9	211	8.0
795	10.6	474	9.5
188	7.8	474	9.5
1073	11.3	868	10.8
387	9.1	330	8.8

You fit the above data to $y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \beta_4 x^4 + \epsilon$. The following output is obtained using Python's statsmodels and sklearn.preprocessing modules.

```
Results                            OLS Regression Results
==============================================================================
Dep. Variable:                          y   R-squared:                       1.000
Model:                                OLS   Adj. R-squared:                  1.000
Method:                     Least Squares   F-statistic:                3.949e+06
Date:                    Fri, 11 Nov 2022   Prob (F-statistic):           8.88e-34
Time:                            10:53:29   Log-Likelihood:                 2.6274
No. Observations:                      16   AIC:                             4.745
Df Residuals:                          11   BIC:                             8.608
Df Model:                               4
Covariance Type:                nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
```

const	1578.3636	708.353	2.228	0.048	19.288	3137.439
x1	-652.5198	299.460	-2.179	0.052	-1311.627	6.587
x2	103.8727	47.219	2.200	0.050	-0.056	207.801
x3	-7.9832	3.292	-2.425	0.034	-15.229	-0.737
x4	0.3050	0.086	3.562	0.004	0.117	0.494

Omnibus:	0.079	Durbin-Watson:	1.818
Prob(Omnibus):	0.961	Jarque-Bera (JB):	0.257
Skew:	-0.128	Prob(JB):	0.879
Kurtosis:	2.434	Cond. No.	1.28e+08

(a) Write down the equation of the fitted curve.

(b) Write down the Python codes to obtain the above results.

(c) Is the model $y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \beta_4 x^4 + \epsilon$ significant for estimating y? Justify your answer using p-value.

(d) Write down the Python codes to determine the predicted value of the mean of y when $x = 7.7$.

(e) Write down the Python codes to created a scatter plot of the data with the fitted polynomial line added to the scatterplot.

Sol:

(a) $\hat{y} = 763.9508 - 316.40x + 52.0456x^2 - 4.4072x^3 + 0.2207x^4$

(b)
```
import numpy as np
import statsmodels.api as sm
from sklearn.preprocessing import PolynomialFeatures
y=[1030,429,429,348,795,188,1073,387,633,498,987,211,
   474,474,868,330]
n = len(y)
x = np.array([11.2,9.3,9.3,8.9,10.6,7.8,11.3,9.1,10.1,9.6,11.1,8.0,9.5,9.5,10.8,8.8]).re
x = PolynomialFeatures(degree=4, include_bias=True)
.fit_transform(x)
Mod = sm.OLS(y,x)
Results = Mod.fit()
print("Results",Results.summary())
```

(c) Since p-value = 2.805537540424102e-34 < 0.05, The null hypothesis of the model not significance for estimating y is rejected. Hence the model $Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + \beta_4 X_4 + \epsilon$ is significant for estimating y.

(d)
```
Beta = Results.params
xh = np.array([1,7.7,59.290000000000006,456.533, 3515.3041000000003])
yh = np.inner(xh, Beta)
print(yh)
```

(e)
```
#create scatterplot
dfx = [11.2,9.3,9.3,8.9,10.6,7.8,11.3,9.1,10.1,9.6,11.1,8.0,9.5,9.5,10.8,8.8]
plt.scatter(x, y)
```

```
Moda = np.poly1d(np.polyfit(dfx, y, 4))
polyline = np.linspace(1, 12, 50)
#add fitted polynomial lines to scatterplot
plt.plot(polyline, Moda(polyline), color='green')
```